A revolution is taking place within the information revolution. It involves harnessing hundreds, thousands, or even millions of computers to work together on a single task.

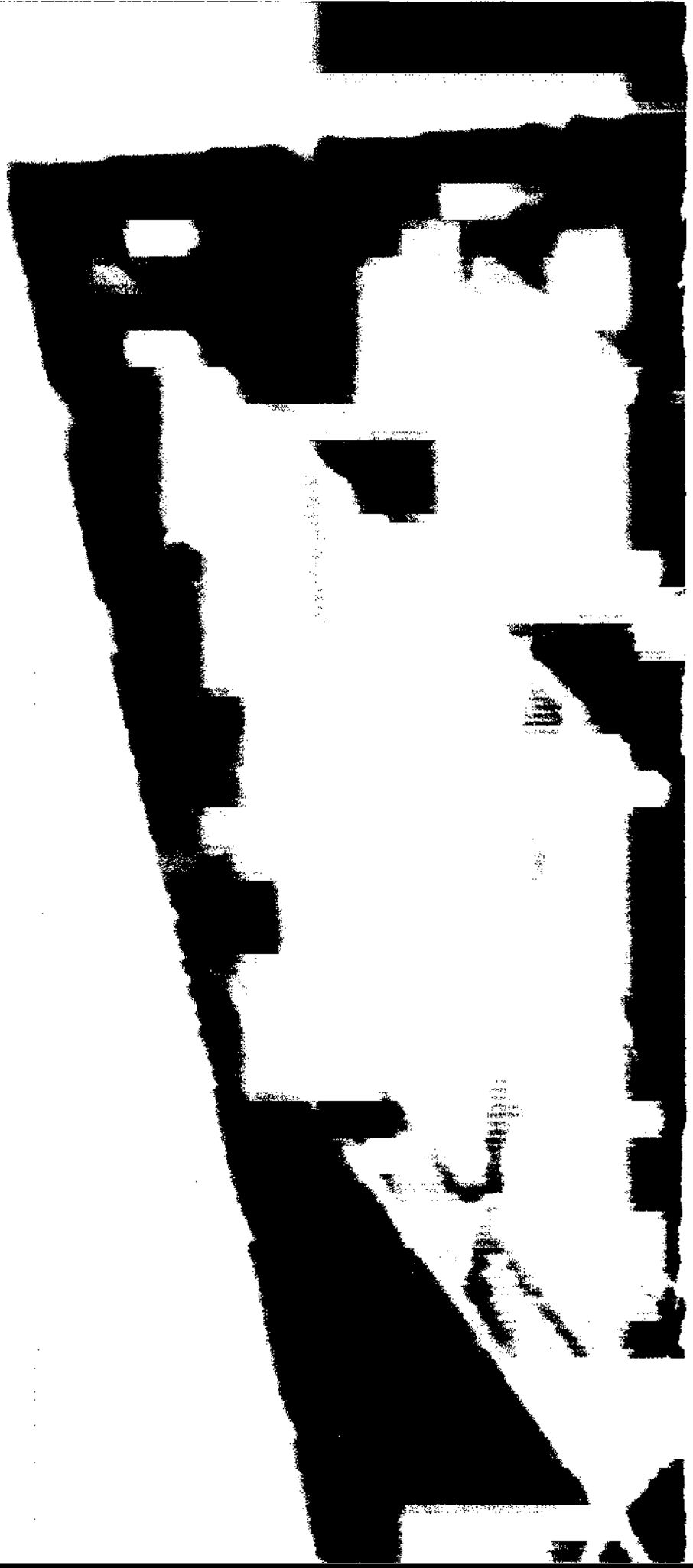
The technical and economic forces spurring this revolution arise from two important trends. On the one hand is the dramatic improvement over the past two decades in very-large-scale-integrated (VLSI) circuits, which now incorporate some hundred-thousand components on a silicon chip. The cost of computational power on chips has dropped an average of 30 percent annually. For less than $100 manufacturers can now fabricate a microprocessor—the computational core of a computer etched on a single chip—capable of performing more than a million instructions per second.

On the other hand, it is becoming disproportionately expensive to construct a single processor, using a number of chips or other components, that can perform calculations appreciably faster than a microprocessor. The world's fastest computers, with only 500 times the performance of a microprocessor, cost 200,000 times as much. Moreover, these large machines cannot be

---

New computing machines could not only be much faster and cheaper, but could achieve the elusive goals of recognizing images, understanding speech, and exhibiting more intelligent behavior.

ILLUSTRATION: TOM NORTON

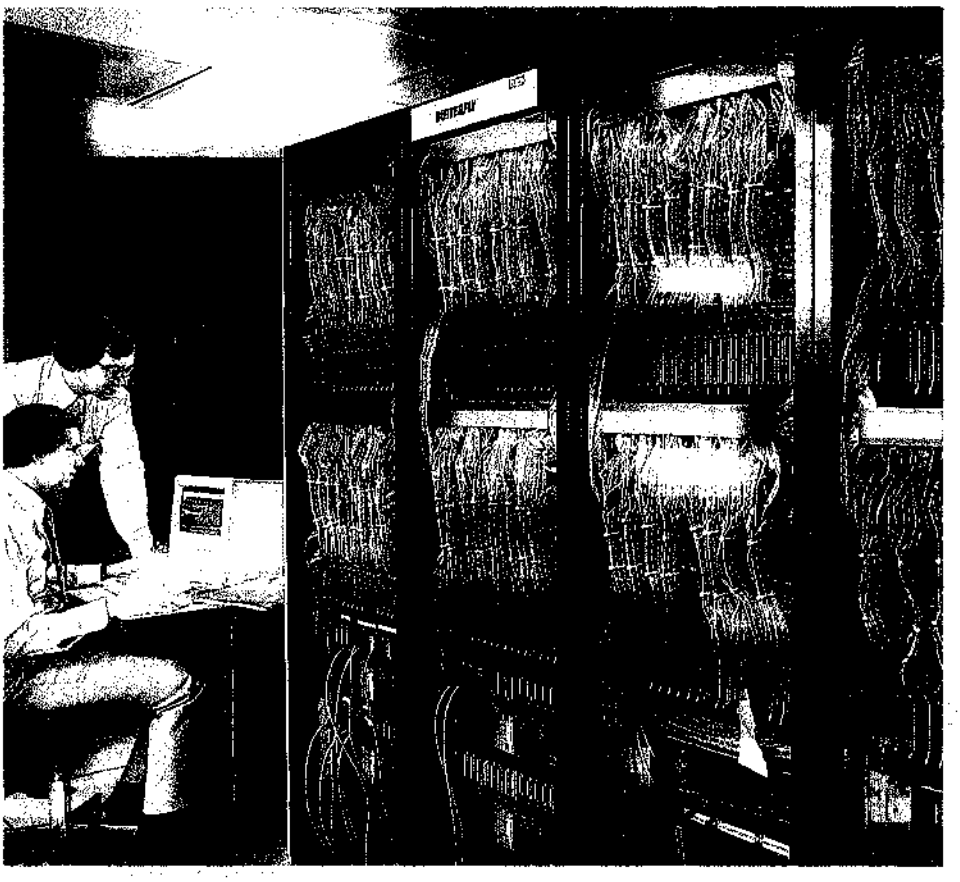


The Multiprocessor Revolution:

HARNESSING COMPUTERS TOGETHER

BY MICHAEL L. DERTOUZOS

Bolt, Ne bridge multip speed While puter that p tions, 128 p work a sing proce the co fast c there limit t tiproc



improved much further because they are pushing against the ultimate limit: the speed of light. They already perform one instruction every 2 billionths of a second, but because electrical signals cannot travel faster than light, information takes at least that long just to move a couple of feet from one part of the machine to another.

By harnessing many relatively inexpensive VLSI processors together into a multiprocessor system—sometimes also called a parallel- or concurrent-processor system—we may significantly reduce the cost of achieving today's fastest computing speeds. More important, there is no foreseeable limit to the speeds we may be able to achieve through multiprocessing. Many of us harbor expectations that this new breed of machines will make possible some of our most romantic and ambitious aspirations: with appropriate programs, these new machines may recognize images, understand speech, and behave more intelligently.

Even anthropomorphic evidence suggests that if computers are to perform intelligently, many processors must work together. Consider the human eye, where millions of neurons in the retina, optic nerve, and visual cortex cooperate to help us see. Or consider the cerebellum, where over a trillion cells share the complex task of higher-level thinking in yet unknown ways. What arrogant reasoning led us to believe that a single processor capable of only a few million instructions per second could ever exhibit intelligence?

To my thinking, the massive quantitative increase in computing power promised by the multiprocessor revolution is a prerequisite to the much-touted qualitative increase in the intelligence of future computers. Of course, merely throwing lots of computing power at difficult artificial-intelligence problems will not necessarily solve them. Otherwise, the use of superfast single-processor computers would have already yielded more progress in this area. But establishing a technology in which an unlimited number of processors can cooperate effectively sets both a machine foundation and a mind set for tackling these difficult problems in a new way. It is my hope and expectation that the additional ingredients we need to achieve greater artificial intelligence—further study of biological intelligence and development of new models for machine vision, hearing, and intelligence—will flourish on this new multiprocessor foundation.

Several dozen research groups in universities and industry are exploring multiprocessor systems. A few computers on the market already use several cooperating processors to solve large problems in high-

PHOTOGRAPH: LITTLEWOOD COMMUNICATIONS

What arrogant reasoning led us to believe that a single processor could ever exhibit intelligence?

energy physics, aerospace engineering, and computer-aided design of products such as computer chips and airplanes.

The promise of the multiprocessor revolution entices and seduces researchers. The outcome of our efforts depends on developments in the three crucial areas that I will discuss: designing architectures for multiprocessors, programming them, and applying them. The value of multiprocessor systems will ultimately rest on our ability to tackle bigger problems simply by using more processors. This "computation by the yard" approach may open the door to a new and more useful world of information technology.

Uses of Multiprocessors

Some traditional information-processing problems require enormous number-crunching power—solving the equations required to predict the weather, calculate the effects of nuclear explosions, or solve certain quantum-mechanics problems. Multiprocessors are likely to be useful for such applications because these problems can typically be decomposed into many interlinked subproblems that can be handled concurrently. For example, a large calculation to predict weather changes over a 1,000-square-mile area could be broken down into 100 smaller calculations, each covering a 10-square-mile cell. These smaller calculations could be performed concurrently by 100 processors. Periodically, the processors would communicate with one another about the weather changes at the boundaries of their cells. Of course, handling these number-crunching applications more effectively, while important, does not constitute a multiprocessor revolution, since their effect on our lives has been and will probably continue to be modest.

On the other hand, replacing mainframe computers with multiprocessor systems that are even slightly cheaper could have a significant economic impact in banks, airlines, insurance companies, and other large organizations. And the prospects for such a change are good, since many business tasks can be broken down into relatively independent subtasks. For example, calculating and printing checks for company employees could be done by many processors working concurrently and almost independently. The processors would need to communicate with one another only about total payroll, tax, and benefit balances. In banks, multiprocessor systems could assign each account to some processor alphabetically or by account number. Concurrent deposit and withdrawal transactions would then be handled by the responsible processors. Airlines might break up centralized programs now used to book reservations, issue tickets, compute fares, and balance cargo. The work could go to several adjacent, communicating machines, corresponding perhaps to each major metropolitan area.

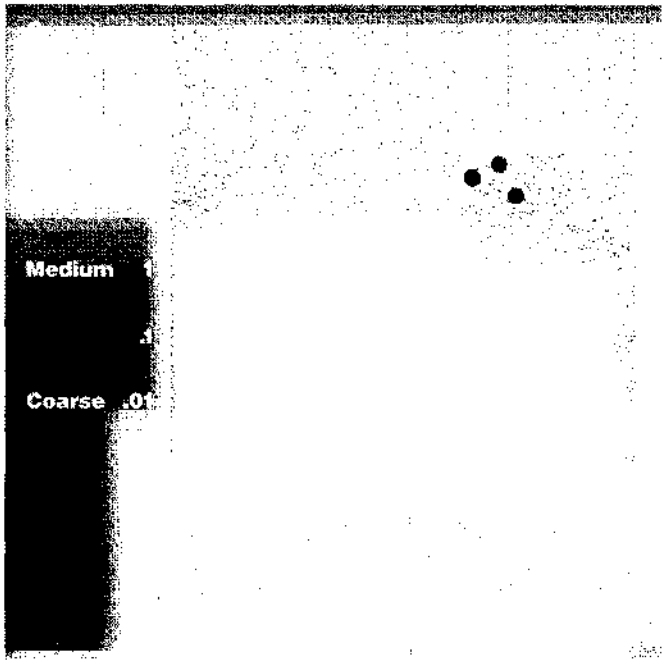
But the revolutionary promise of multiprocessors rests with their expected ability to achieve what is difficult or impossible for conventional computers—artificial intelligence (AI) applications involving functions such as seeing, hearing, learning, reasoning, and advising.

To understand the utility of multiprocessors better, I recently convened an M.I.T. workshop on the prospective usefulness of multiprocessors. The participants included some 25 leading researchers who are applying computers to a variety of fields, including medical diagnosis, aerodynamic design, and speech recognition. I asked them how many calculations could be carried out concurrently on future multiprocessor systems to solve problems in their areas. These specialists had to imagine how they might decompose operations now performed sequentially into relatively independent subtasks that processors could, with some intercommunication, perform concurrently. This mental task is difficult and cannot be done precisely, but it is not as tough as it sounds. It is similar, in terms of mental gymnastics, to what programmers must do today in decomposing a large task into subtasks to be executed in sequence, and into "utility subroutines"—subprograms executed frequently to carry out many of the higher-level tasks.

I also asked these researchers to identify the most desirable "granularity" of multiprocessor systems for their applications. Granularity is a measure of the size and complexity of the processors in a system.

MICHAEL L. DERTOUZOS is professor of computer science and electrical engineering at M.I.T. He is director of M.I.T.'s Laboratory for Computer Science, which developed time-shared computers and has pioneered in multiprocessor research over the last 15 years. He steered the LCS into this area and helped the Defense Advanced Research Projects Agency launch its multiprocessor program. A member of the group that founded the Microelectronics and Computer Technology Corp., he got this consortium of computer and semiconductor firms interested in multiprocessor research. Some of his prior research has been in personal computers, and in a People Magazine *interview 11 years ago, he predicted the revolution that later occurred in this field.*

Fine-grain systems have many simple processors (each chip may contain 100). Coarse-grain systems have fewer, powerful processors (each may contain 100 chips). Specialists at an M.I.T. workshop estimated the optimal number and granularity of processors for their applications. (Numbers in parentheses represent different researchers' estimates.) Sensory applications may use millions of fine-grain processors; deeper, cognitive applications may use hundreds of coarse-grain processors.



A fine-grained multiprocessor system has small, simple processors, but generally many of them. By my somewhat arbitrary definition, in a fine-grained multiprocessor, a single silicon chip has some 100 processors on it. A system with 10,000 chips would have 1,000,000 processors. A medium-grained system has more powerful processors, say 1 per chip. Such a system might have 10,000 chips and the same number of processors. A coarse-grained system has even more powerful processors—each one might be composed of 100 chips—but it might have only 100 processors in all.

The results of my survey show that tasks such as vision and speech recognition may profitably use millions of fine- and medium-grained processors working concurrently. Deeper cognitive applications and more traditional computing jobs are perhaps better handled through a few hundred coarse-grained processors. This conclusion makes intuitive sense. Thousands or even millions of hair-like cells in our ears and retinal cells in our eyes are concurrently stimulated by a single musical sound or a quick glance through a car window. Yet after these stimuli have been processed by our brains for a while, we are left with a more sequential, conceptual residue—"Aha . . . the opening bars of Beethoven's Fifth," or, "He veered to the right to miss the bicyclist and couldn't avoid the pole." Indeed, the very essence of reasoning or arguing a point seems at a deep conceptual level to be a sequential process. The survey results mirror what I believe is the case with the human cognitive pyramid: the number of simultaneously performed tasks seems to increase dramatically toward the sensory base.

Clever information-processing assistants that can see, hear, and think a little will be a liberating and welcome change from today's machines. Advertising notwithstanding, current computers are dumb and unfriendly: witness the inexcusable 10-inch-thick manuals that often accompany professional word-processing programs. Instead of typing, users of multiprocessors will be able merely to issue verbal commands and show graphs or text to the machine. More important, users will be able to employ a more natural and less precise idiom since, by virtue of their increased intelligence, these new computers will be more forgiving than their allegedly friendly predecessors.

Possible uses of multiprocessor systems beyond speech recognition and machine vision include:

- ☐ Automated clerical assistants.
- ☐ Tutors that can understand students' progress and can ask questions to correct weaknesses.
- ☐ More intelligent retrieval systems that can understand what information the user wants—in finance, law, medicine, government, and other fields.
- ☐ Knowledge-based systems that know a great deal—and can help users—in narrow areas of expertise such as investing and medical specialties.
- ☐ Robots that can perceive their environments and act appropriately—perhaps to do hazardous work such as cleaning up toxic waste or jobs requiring great precision.
- ☐ Monitors that watch or listen to numerous events and flag the most important for the human user to review.

Architectures

Multiprocessor systems should not be confused with the distributed computer networks that are already becoming the backbone of office automation. In these networks, relatively slow local or long-haul connections generally link a number of geographically remote machines, communicating information at a few hundred to a few hundred thousand characters per second. Multiprocessor systems are in one location and communicate data among their processors at a few million to a few hundred million

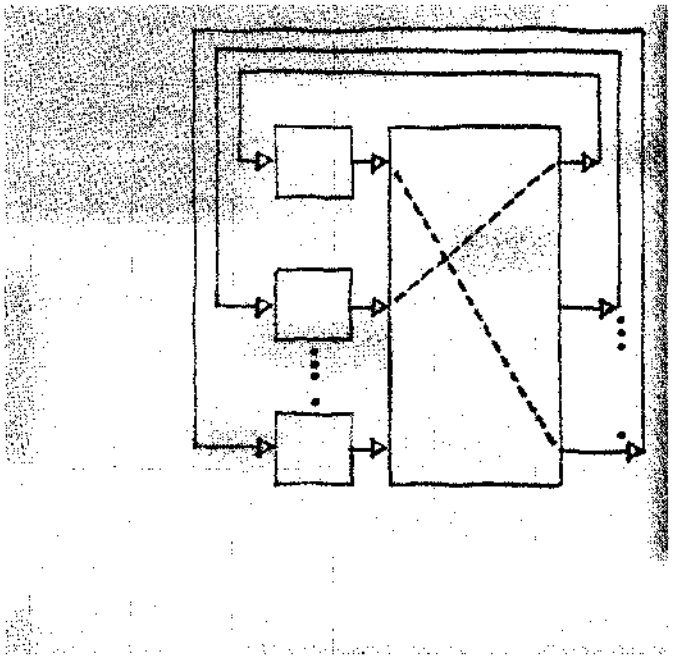
DIAGRAMS: OMNIGRAPHICS

characters per second. More important, a distributed system is a collection of predominantly autonomous machines, each controlled by its own user but able to communicate with other computers—such as in sending electronic mail or requesting data. By contrast, in multiprocessor systems, many processors work under a central control on a single problem. In a loose analogy, a distributed network is like a society of communicating individuals and organizations, while a multiprocessor is like the aggregate of neurons in a single human brain.

Today research groups in some 50 universities, industrial research laboratories, and start-up companies around the world are experimenting with multiprocessors. While it is hard to classify these machines precisely, I will describe a few of them to convey a flavor of the different possible architectures and the problems associated with each.

A typical "dataflow" multiprocessor is the Tagged Token Dataflow Machine that Professor Arvind and his group are building at the M.I.T. Laboratory for Computer Science (LCS). Machines of this sort are relatively coarse-grained—our machine at LCS has 50 to 100 processors. A communication network like a railroad switchyard connects these processors. Every time a processor produces a unit of data, it incorporates a "tag"—information on where the data should be sent and how they should be used. The "switchyard" uses part of these tags to route the data to another processor. Once there, the data wait for any related data that may be required for the next computation. When all the required data have arrived, the new processor acts upon them, following procedures designated by the tags. This processor sends the results—new data, with new tags—back to the network, where the whole process repeats itself. In a properly working dataflow machine, almost all the processors should be humming away, issuing tagged data that are switched to other processors through the network.

In a "tagged token dataflow" machine, processors issue data with "tags." Like a railroad switchyard, the communication network uses part of each tag to route data to the processor that will perform the next operation. Another part of the tag tells this new processor what to do. In a properly working dataflow machine, almost all of the processors should be continually humming away, issuing tagged data that are switched to other processors through the communication network.



Invented at the M.I.T. Artificial Intelligence Laboratory and under construction by Thinking Machines Corp., the Connection Machine typifies another approach. The first model of this machine consists of 65,536 fine-grained processors connected to one another in three ways. First, the processors are arranged on a grid, where each communicates with its neighbors on four sides—the "North," "South," "East," and "West." Second, the "broadcast network," a slower communication line, passes through every processor and links all of them to a central control computer. Finally, a "16-dimen-

SOME MULTIPROCESSOR PROJECTS

UNIVERSITIES

California Institute of Technology
Columbia University
Illinois University
Manchester University
Massachusetts Institute of Technology
New York University
Purdue University
University of California, Berkeley
University of London
University of Maryland
University of North Caroline
University of Texas
University of Utah
Washington University

COMPANIES

Alliant
ATT
Axiom
BBN
Control Data
Cray
Culler Scientific
Denelcor
Encore
ETL (Japan)
Flexible Computer
Goodyear
Harris
ICOT (Japan)
Intel
IBM
MCC
Myrias
NEC (Japan)
Sequent
Thinking Machines

sional-hypercube network" also connects the processors.

To visualize such a hypercube, start with a two-dimensional "hypercube"—an ordinary square. A square has four corners, or vertices. To make a three-dimensional hypercube (i.e., an ordinary cube), place one square in front of another and connect the corresponding vertices of the two squares with lines that form four new edges. Now suppose there is a processor at each vertex and a communication line along each edge. That makes eight (2^3) processors in all. Any of these eight processors can communicate with any other by sending a signal through no more than three communication lines.

Now repeat the geometric construction process using two three-dimensional cubes. In other words, juxtapose the two cubes diagonally and connect the corresponding vertices with eight new edges (diagonal lines). The result, which can be drawn in ordinary three-dimensional space, is what we will call a four-dimensional hypercube. It, too, can be thought of as having a processor at each vertex and a communication line along each edge. But since it is composed of two three-dimensional cubes, it has twice as many processors—16 (or 2^4). Any processor can communicate with another by sending a signal through no more than four communication lines.

Similarly, you can connect two 4-dimensional hypercubes to get a 5-dimensional hypercube, and you can extend the process to as many dimensions as you wish. A 16-dimensional hypercube has 65,536 (or 2^{16}) processors, and each processor can communicate with any other by sending a signal through at most 16 communication lines. The hypercube network is used in the Connection Machine and other multiprocessors because it allows many processors to communicate relatively quickly using a reasonable number of wires. The alternative of connecting every processor to every other would result in an impossible wiring task, with over 2 billion wires.

The Connection Machine is a fine-grained multiprocessor. Each processor is quite small—having around a thousandth the complexity of a processor in a dataflow machine such as Arvind's—and is capable of processing fewer instructions on smaller chunks of data. The central computer initially feeds the processors with data through the broadcast network. The processors can then repeatedly compute and send data to their North-South-East-West neighbors, to their neighbors on the 16-dimensional hypercube, or back to the central computer.

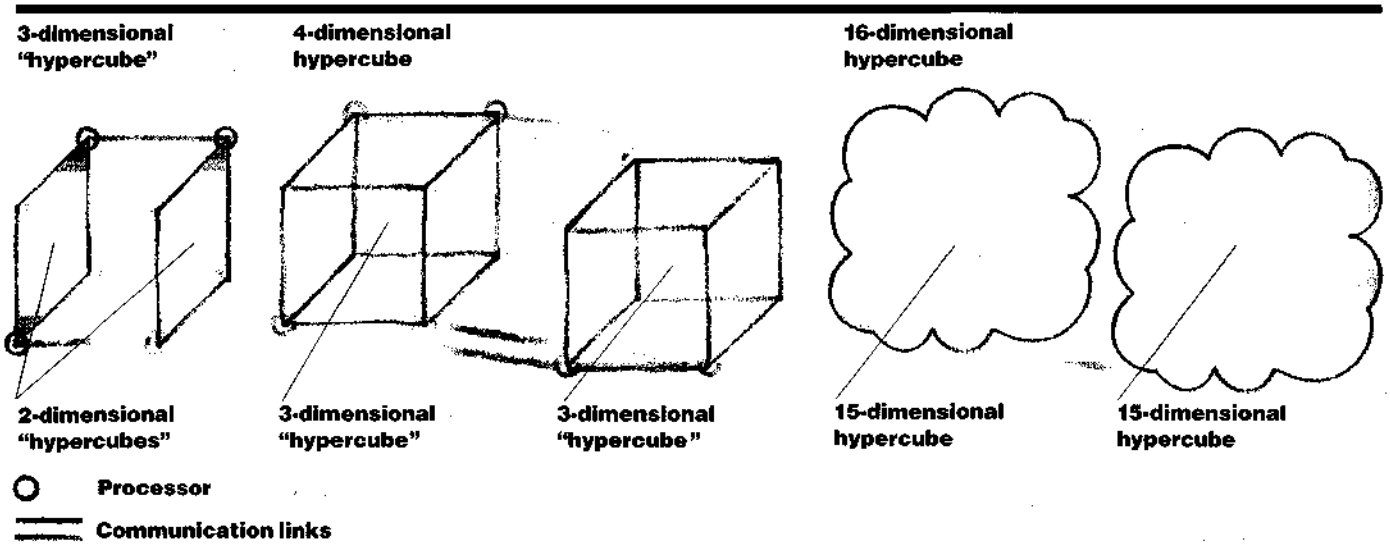
Machines like this will probably be especially useful in sensory applications, where fine-grained processors and adjacent communications are important. For example, a two-dimensional "retina" could be made to recognize lines or edges when adjacent cells—i.e., adjacent processors—register a common light value along a particular direction. To recognize such lines or edges, processors would communicate on the North-South, East-West grid. After identifying such lines and edges, the multiprocessor system could use them to build an abstract, higher-level model of what the retina "sees." Building such a model could involve communication among more distant processors on the hypercube.

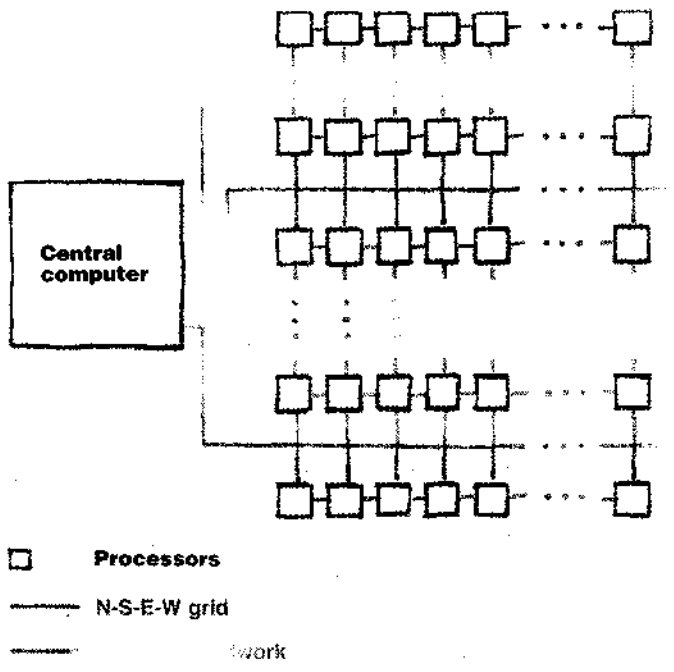
Perhaps the most common multiprocessor architecture uses a "shared memory" to communicate information among processors. In this scheme, each processor resembles a conventional computer with a local memory and peripheral devices such as sensors, printers, and screens. A memory common to all processors serves as a blackboard: any processor can write information on it, which any other processor can subsequently read. If processor A wishes to communicate with processor B, it writes its message on the shared memory, and B reads it. However, while the writing and reading is going on, no other processors can communicate. The common memory thus becomes a bottleneck. To compensate for this limitation, the shared memory is typically very fast and is often augmented with auxiliary wiring so that the system can accommodate several tens to several hundreds of communicating processors.

Shared-memory architectures are relatively inexpensive and represent a conservative approach, since they permit a gradual transition from today's single-processor, single-memory computers to multiprocessor systems. Companies such as Flexible Computer Corp. of Dallas use machines with this kind of architecture, marketing them for aerospace and scientific applications.

Programming Multiprocessors

There are as many different approaches to programming multiprocessor systems as there are to organizing a multiprocessor architecture. In the "functional-language" approach, the programmer uses instructions that behave like mathematical functions—taking particular inputs and producing out-





In the Connection Machine, numerous processors are connected to one another in three ways.

First, a relatively slow "broadcast network" links a central computer to every processor. Second, the processors are arranged on a "North-South-East-West" (N-S-E-W) grid. Each communicates with its neighbors on four sides—"North," "South," "East," and "West." Finally, a "16-dimensional hypercube" (not indicated here) connects the processors.

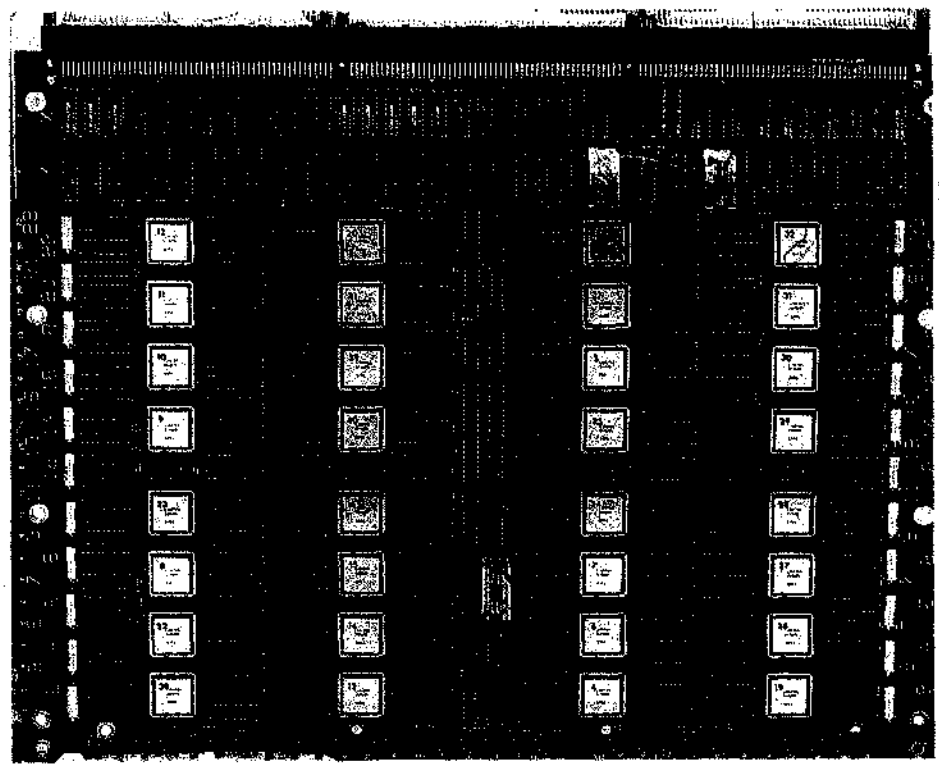
Top: Illustration of the geometric principle behind the hypercube (explained on page 50). The hypercube allows each processor to communicate with any other through no more than 16 communication lines. Machines like this with numerous, relatively simple processors and adjacent communications might be especially useful in sensory applications. For example, such a machine could work like a simple retina.

puts. One elementary function, PROD, might find the product of inputs. For example, PROD (3,10,4,5) would have four inputs and one output—in this case, $3 \times 10 \times 4 \times 5$, or 600. Another function, SQ, might square its input if it is between 1 and 1,000 but leave it unchanged if it is outside this range. Thus, SQ (5) would be 25, and SQ (2,000) would be 2,000.

Of course, functional languages use functions to perform operations that are far more complex and useful than these simple examples suggest. More important, functional languages allow basic functions to be combined into aggregates, in which the outputs of some functions become inputs to others. These aggregates themselves behave like functions, in that they take inputs and produce outputs. Thus, they can be combined into yet more complex aggregates, which also behave like functions. Functional programs can be thought of as huge diagrams of interconnected functions, in which the outputs of one function are inputs to another, until they produce one or more final results.

Significantly, functional programs like these have only inputs and outputs and no "side effects." Side effects are operations that store certain results in memory "on the side" while a program processes inputs to produce outputs. Thus, a procedure may take an input and square it to produce the output, but may simultaneously increase a running total in memory by the same amount as the output. Completing some later step in the program will require referring to this running total. Today's languages encourage the use of side effects, and nearly all practical programs employ this device.

Functional languages may be harder to use, but



One of the circuit boards from the Connection Machine, a multiprocessor system developed by Danny Hillis of Thinking Machines Corp. of Cambridge, Mass. Each of the 32 chips plugged into the board contains 16 processors. Thus, the board has 512 processors. The Connection Machine will have 65,536 processors in all.

they have several advantages. They result in programs with cleaner structures, precisely because there are no confusing side effects to keep track of. Moreover, one aggregate function can easily be replaced by another that produces the same outputs from given inputs, but that has a simpler internal structure and is therefore faster to execute. With the functional approach, it is easier to verify that large programs operate correctly, a task that is often difficult with today's programs. And functional programming makes it easier to specify a program—to give a precise, brief description of everything it is supposed to do before the programmer starts to write it.

In the functional-language approach, the programmer can be fairly oblivious to the physical structure of the machine and is liberated from the task of assigning operations to particular processors. Instead, another program, a multiprocessor compiler, automatically splits the primary program into concurrent subprograms for different processors and specifies the necessary intercommunications. This permits the programmer to focus on the functions to be done rather than on coordinating individual computations. However, this approach may lead to programs that are inefficient to execute. The compilers that establish which operations should run concurrently may miss shortcuts and opportunities that programmers could specify.

In what I would call the "concurrency-extension" approach to programming, a minimal set of instructions added to conventional computer languages such as Fortran, C, or Lisp enables the programmer to specify which processes should run concurrently. Extended languages of this sort have names like Pardo (parallel do) Fortran, Concurrent C, and MultiLisp. Programs written with this approach are generally efficient to execute because programmers can use their ingenuity in specifying which operations should run concurrently. But coordinating large numbers of processors may take the programmer a lot of time. Shared-memory architects often use concurrency extension because it is similar to today's single-processor languages, involving only one or two new instructions.

To get a clearer idea of how multiprocessor programs work, consider a very simple problem. Let a, b, c, and d be any four numbers, and suppose you want to calculate ac + ad + bc + bd. (As in customary algebraic notation, ac indicates the product of a and c.) Assume furthermore that each processor can handle only two numbers at a time and that it takes one microsecond for addition (+) and five microseconds for multiplication (×).

A conventional single-processor computer would carry out the necessary additions and multiplications sequentially. If T_1, T_2, and so on are intermediate results that need to be operated on further to produce the final result, the calculation might be done this way:

PHOTOGRAPH: THINKING MACHINES

The author at M.I.T.'s Multiprocessor Emulation Facility, where up to 100 machines can be connected in different configurations by high-speed switches. This "computational sandbox" makes it possible to simulate multiprocessor designs inexpensively. Before constructing a 1,000-processor "retina," a researcher could use the facility to emulate the architecture, program it, test it, and make improvements.

Sequential Computations	*Time (microseconds)*
$T_1 = a \times c$	5
$T_2 = a \times d$	5
$T_3 = b \times c$	5
$T_4 = b \times d$	5
$T_5 = T_1 + T_2$	1
$T_6 = T_3 + T_4$	1
Answer $= T_5 + T_6$	1
	23 *Total*

The total time to perform the computation this way is 23 microseconds.

In a multiprocessor system programmed in a functional language, you need to break the problem down into a set of elementary functions and directions for combining them. Let "SUM" be the addition function—the output of SUM (2,5,7) is the sum 2+5+7, or 14. Let "PROD" be the multiplication function—the output of PROD (3,7) is the product 3×7, or 21. The process of finding ac + ad + bc + bd can be expressed in functional terms as:

SUM (PROD (a,c), PROD (a,d), PROD (b,c), PROD (b,d))

Given this functional expression, the compiler might automatically apportion the calculation to several processors, as shown below, where concurrent operations are on the same line:

Concurrent Computations	*Time (microseconds)*
$T_1 = a \times c \quad T_2 = a \times d \quad T_3 = b \times c \quad T_4 = b \times d$	5
$T_5 = T_1 + T_2 \quad T_6 = T_3 + T_4$	1
Answer $= T_5 + T_6$	1
	7 *Total*

Doing this calculation with four processors operating concurrently takes only 7 microseconds—practically a third the time it takes with only one processor.

In a language where the programmer can specify concurrency explicitly, you might do the operation another way. If you recognize the algebraic identity

$$ac + ad + bc + bd = (a + b)(c + d)$$

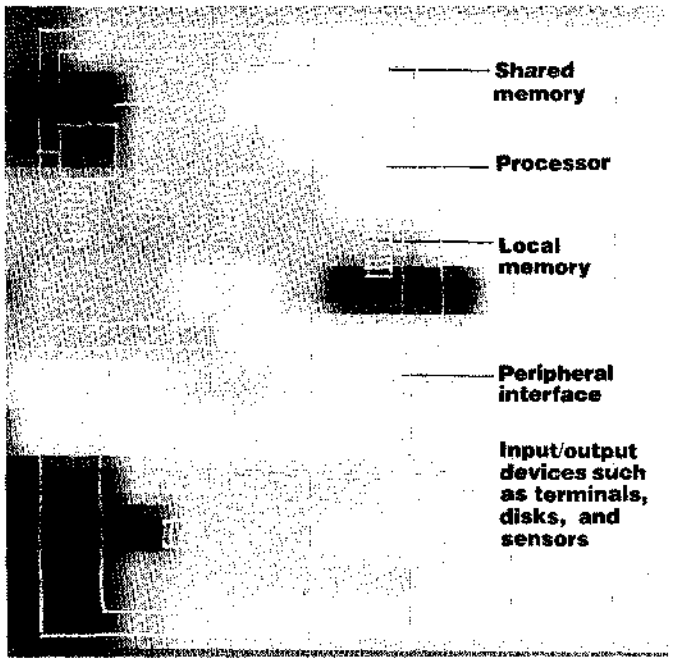
you can write the following:

Concurrent Computations	*Time (microseconds)*
Do concurrently ($T_1 = a + b$, $T_2 = c + d$)	1
Answer $= T_1 \times T_2$	5
	6 *Total*

This would take only 6 microseconds to execute and would require only two processors—the most effi-

PHOTOGRAPH: FRANK SITEMAN

Shared-memory architectures such as this permit a gradual transition from today's single-processor computers to multiprocessor systems. Each processor resembles a conventional computer with a local memory and peripheral devices such as sensors and printers. A common memory serves as a blackboard: any processor can write information on it, which any other processor can then read. Shared-memory machines are already marketed for aerospace and scientific applications.



cient calculation yet in terms of time and computer hardware.

Proponents of the functional approach would argue that the compiler spares programmers the clerical work of deciding which operations to do concurrently. Proponents of allowing the programmer to specify concurrency explicitly would counter that allowing the programmer to specify concurrent calculations saves computing time. Advocates of the functional language approach might respond that it could achieve the same computing efficiency if the programmer specified in the first place:

Answer = PROD (SUM (a,b), SUM (c,d))

Researchers will undoubtedly continue to debate the key question of how work should be apportioned between automatic compilers and human programmers.

Yet another programming approach is based on the belief that we will be able to run today's single-processor programs on tomorrow's multiprocessors. Some computer scientists are attempting to achieve this goal by developing so-called "unraveling compilers" that take as input a conventional Fortran program and generate as their output an equivalent program for a particular multiprocessor. This approach may prove feasible for small multiprocessors but not for larger machines, since the structure of today's programs is inextricably bound to the single processors that we use. To expect that efficient multiprocessor programs can be automatically extracted from single-processor programs is tantamount to hoping that the procedures of a nineteenth-century cobbler could be automatically transformed into specifications for a modern shoe factory with thousands of workers.

To my thinking, researchers must invent a whole family of new languages for multiprocessor applications. I envision that these languages will automatically detect and handle relatively simple, lower-level tasks that can obviously be executed concurrently. But for higher level, more complex operations, these languages will permit programmers to specify explicitly what should be done concurrently and how processors should communicate. The model I have in mind is like a huge data factory, full of bins that store data, conveyor belts that transport data, and assembly lines that process data. The programmer would tune this data factory, much as industrial managers tune actual factories to optimize production.

Finally, I believe that, regardless of the method used, programming multiprocessor systems will inevitably require more effort than programming today's single-processor systems, since the programmer will have to specify and coordinate so much more.

Thinking Concurrency

Since computer science is a predominantly experimental discipline, we can best design a new multiprocessor after simulating it on a computer, where we can adjust parameters and assess performance. However, such a simulation requires a huge number of concurrent activities that cannot reasonably be modeled by a conventional, single-processor computer. Thus, we can effectively simulate a multiprocessor only on a multiprocessor—a circular situation. Such problems are typical in computer science. In this case, we take our best guess at designing an initial multiprocessor. Then we bootstrap our way up by using this machine to simulate possible new multiprocessor systems.

This is the idea behind the Multiprocessor Emulation Facility at the LCS. This facility consists of 50 to 100 machines connected by high-speed switches,

In designing multiprocessors, we could learn from our own bodies, which handle numerous failures effectively.

which can be used to create many different configurations of interconnections. For example, we can create a North-South, East-West grid; a ring; or an eight-dimensional hypercube. We plan to use this Emulation Facility like a computational sandbox, where we can try out alternative designs for much larger and different multiprocessor systems. A researcher with a bright idea for constructing a 1,000-processor retina could use the facility to emulate that architecture, program it, test it (at a fraction of its ultimate speed and cost), and make improvements.

This multiprocessor sandbox allows us to "think concurrency." We believe that such thinking will allow us to invent richer languages and architectures than we could if we used only paper and pencil—or if we simplified multiprocessor architectures to simulate them on single-processor machines. The latter approach is as effective as trying to perceive a rich image by sliding over it a piece of paper with a peephole.

Reliability and the Swiss-Watch Syndrome

One of the drawbacks of a computer with many processors is the increased opportunity for both hardware and software malfunctions. Today's single-processor machines can run without a major hardware fault for up to a few thousand hours. Software errors can take place within tens or hundreds of hours. At such rates, a 1,000-processor system would have a hardware failure every few hours and a software failure every few minutes. We cannot allow a single failure to cause an entire multiprocessor system to crash.

Some multiprocessor designers try to incorporate error detection and correction into their architectures. For example, the processors may check their own soundness or that of other processors by running small test procedures. While the program is being executed, the system assigns computations only to those processors that are in good running order. Such a multiprocessor is analogous to a factory in which management assigns the jobs of sick employees to healthy ones.

We still have little practical experience in detecting and correcting errors in large computer systems, and some researchers are postponing work on this problem until they solve what they feel are more pressing architectural and programming issues. I believe this is a misguided approach, since reliability may turn out to be pivotal to the evolution of multiprocessors. Today nearly all of our computers and their programs are like precision watches—one small flaw on one small gear and the whole mechanism grinds to a stop. In designing multiprocessors, we could learn a great deal from our own bodies, which can handle numerous failures effectively.

I believe we can accomplish this only through radical architectural innovations based on notions of "continuity" and "mushiness," so that small changes in one part of the system cause only small changes in every other part. In imitation of the human nervous system, we might compromise between digital and analog modes of encoding data. Most computers record information digitally in a collection of on or off signals; the sound-wave forms on conventional LP records are analog. Digital recording can be as accurate as required, but a little damage can result in catastrophic change in the recorded data. Comparable damage to an analog recording is relatively minor: a slightly scratched record still reproduces the song. To incorporate the benefits of digital accuracy and analog tolerance to faults, information could be encoded as standard electrical pulses that have the same size but are emitted at different rates. The rates could vary continuously, so that a small error in one signal or in the system would only cause a small distortion in the output. But such "mushy" systems may have to wait for yet another revolution beyond the precision-oriented multiprocessors that we are working on today.

The Global Race

Only a few U.S. research laboratories pursued multiprocessing until 1980, when Japan announced its Fifth Generation Computer Project. That effort, involving some $1 billion over 10 years, aspires to tackle AI problems—to translate automatically between Japanese and English and devise programs that embody human expertise. To achieve these ambitious goals, the Japanese are pursuing multiprocessor machines intensely. The Institute for New Generation Computer Technology (ICOT) spearheads this research and apportions it to participating companies. Having established an advanced electronics infrastructure, the Japanese are now trying to become familiar with the looser, more tentative research approach required for AI.

The United States has acted—or reacted—in sev-

By skewing research toward immediate military objectives, the U.S. may cede leadership to Japan.

eral ways. The Microelectronics and Computer Technology Corp. (MCC), a consortium of leading U.S. semiconductor and computer firms, is financing and sharing research of common interest to all the firms. Multiprocessor systems are the most important of MCC's four research areas.

On a more traditional front, the Department of Defense's Advanced Research Projects Agency (DARPA) has launched the Strategic Computing Program. Multiprocessor research receives a major part of the program's $150 million annual budget. Also, U.S. venture capitalists have begun to finance companies that build multiprocessors. At this writing, there are over ten such start-ups, with about half still doing R&D and the other half starting to sell products. Big companies such as IBM have cautiously begun advanced research in this area, too.

Europe has launched at least two multiprocessor programs. The British Alvey effort, with fewer funds than Japan's, focuses primarily on forefront computer research and on the kind of software necessary for AI applications. The European Economic Community (EEC) has launched ESPRIT, a program with funding comparable to the Japanese effort. ESPRIT spans many areas of information technology and involves many countries.

If Japan reaches its immediate objectives, it may indeed approach its ultimate stated goal—world supremacy in information technology. Even if it does not reach this goal, the country may gain substantial benefits. For example, the Japanese could develop products with capabilities for vision, speech, and intelligence—such as a voice-controlled television set or a car that can diagnose its malfunctions.

On the other hand, the United States stands a good chance of retaining its leadership in information technology if it pursues the multiprocessor challenge. We need only use the proven mechanisms of DARPA-funded programs and venture capital, which have been responsible for the landmark innovations in information technology. DARPA-funded research gave rise to time-shared computers, artificial intelligence, expert systems, and computer networks, while firms funded by venture capital developed the microprocessor and home computer.

The relative U.S. disadvantage in production—lacking highly organized, coherent teams of workers that follow precise objectives—is our primary advantage in innovation. Forefront research in computer science likes a flexible, experimental, exploratory approach and does not react well to detailed management or preset objectives.

However, several recent, disturbing trends could impair just this flexible approach and hence U.S. success. The government has become increasingly preoccupied with possible leaks of research to military adversaries and commercial competitors. The government has also begun to seek research results with more immediate military relevance, and it has instituted increasingly lengthy and bureaucratic processes for allocating funding to new projects, akin to the competitive bidding for weapons contracts. These trends reflect a well-intentioned desire to tighten the government's management of basic research. However, because of the freedom that such research requires, this trend toward increased management could do far more to damage innovation than to accomplish the intended goals. As a result, the United States could cede leadership in information technology to Japan.

The EEC effort, spread thin by the large number of participants and the inevitable political orientation of a consortium of nations, is not yet a major contender in multiprocessor technology. However, the EEC effort could surprise us because of the strength of European theoretical researchers.

Technological leadership of the information revolution is, in my opinion, ultimately linked to geopolitical strength. At the turn of this century, England and the other leaders of the industrial revolution held the reins of the world because of their industrial might and knowhow. In the forthcoming era of a service-dominated economy, information will undoubtedly play a progressively greater role in all facets of our professional and personal lives. The nations that master this new revolution will exert leadership in the economic and geopolitical arenas.

We are not yet sure whether multiprocessors will offer as revolutionary a jump as we envision in the use of computers. Nor are we certain whether they will help us move much closer to the elusive and seductive goal of greater machine intelligence. The prospects look good, and the emotional temperature is high with expectation. But we must wait for the results of many experiments now in progress before we can give a more unbiased and factual assessment. The answers may well come within five years. Meanwhile, we can continue to dream, hope, and work toward these new and exciting prospects in information technology. □